APPENDIX A
PSEUDO-CODE EXAMPLE

```
/
/ This is a sample pseudo-code program that demonstrates a delayed
/ acknowledgement mechanism for implementing fault tolerance.
/ Note this is just one representative implementation of the methodology.
/ It assumes the availability of certain middleware capabilities through
/ several hypothetical middleware library calls.
/
/ Authors: Mingqiu Sun and Mahesh Bhat
/ Date: 5/10/2001
/
/*****************************************************************/

#include <comutil.h>
#include <iostream.h>

/*****************************************************************
/ The following class represents callback functions that handle client requests.
/
/*****************************************************************
class EventHandler
{
        App* m_pMApp;
        TargetApplication *targetApp;   // to perform a task in a workflow sequence

public:
        EventHandler(App* pApp)
        { m_pMApp = pApp; }

        virtual ~EventHandler()
        {}

private:
        /************************************************************
        / The following callback function implements a simple workflow sequence that has
        / several tasks: 1. connect to an application; 2. pass the message from client
        / to the application; 3. finish processing and send confirmation back to client.
        /************************************************************
        virtual void onEvent( const Event& refEvent)
        {
                const Event* pEvent = refEvent.getEvent();
                if (pEvent) {
                        String sMessage = pEvent->getMessage();

                        //This corresponds to Task $T_1$ (FIG. 4) where it is trying
                        //to connect to the application.

```

```
                int connected = targetApp.connect();

                //This call corresponds to T2 (FIG. 4).
                //Here it tries to load the message in the application.

                int loaded  = targetApp.loadData (sMessage);

                //This corresponds to final task TFT (FIG. 4) where it checks if
                everything //is fine.
                if(loaded)
                {
                        //Trigger the confirmation
                        int triggered = pEvent.trigger();

                        //if triggered, then send conformation back to the client
                        //this action could be performed by a different process
                        if (triggered)
                                pDataEvent->sendConfirmationBack();
                }
                // else client would resend request back due to a lack of confirmation

            }
        }
}; // EventHandler



/*****************************************************************
/ The following class represents a process that handles incoming client requests. It
/ launches a certified messaging listener on start, which in turn launches a callback
/  function to start a workflow sequence upon arrival of a client request.
/
/*****************************************************************
class MessageProcessor : public App
{
public:
   MessageProcessor( AppProperties* pAppProperties);
   ~MessageProcessor();

protected:
   virtual void onStart() throw(Exception);
   virtual void onEnd() throw(Exception);

private:
   Subscriber*                  m_pMSubscriber;
   DataEventHandler*            m_pDataEventHandler;
}; // MessageProcessor


MessageProcessor::MessageProcessor( AppProperties* pAppProperties)
 :App(AppProperties)
```

```
{
}

MessageProcessor::~MessageProcessor()
{
}

void
MessageProcessor::onStart()
{
        MProperties* pMProps = App::getProperties();

        // subscriber
        MsgSubscriber* pMSubscriber = getMsgSubscriber();

        m_pEventHandler = new EventHandler( this);
        pMSubscriber->addProcessor( m_pEventHandler );
}

void
MessageProcessor::onEnd()
{
        delete m_pEventHandler;
        delete m_pMSubscriber;
}



/*****************************************************************
/ The following is the main program that creates a MessageProcessor
/
/*****************************************************************
int main(int argc, char** argv)
{
        String appName = "testMessageProcessor";
        String appVers = "3.0";

        try
        {
                AppProperties appProps;
                appProps.setAppName(appName);
                appProps.setAppVersion(appVers);

                MessageProcessor* pMessageProcessor = new MessageProcessor(&appProps);


                //This corresponds to the TST process in the workflow sequence.
                pMessageProcessor->start();

                // exiting from main loop - perform cleanup and exit
```

```
                delete pMessageProcessor;
        }
        catch(Exception& e )
        {
                String s = e.getType();
                cerr << s << endl;
        }
        return 0;
} // main
```